VAULT CONTRACT SYSTEM

December 12, 2024

1 Introduction

Zerobase is a real-time zero-knowledge (ZK) prover network designed for rapid proof generation, decentralization, and regulatory compliance. It generates ZK proofs within hundreds of milliseconds, enabling large-scale commercial applications.

We introduce a smart contract system, hereafter referred to as *Vault*, it is a secure staking and rewards management system built on the Ethereum Virtual Machine (EVM) blockchain. It allows users to stake supported tokens and earn rewards.

2 Protocol Architecture

2.1 Features

The *Vault* contract, as a staking product, provides basic deposit and withdrawal functionalities. After staking for a certain period, users earn a corresponding reward based on the reward rate. Generally, the *Vault* has the following features:

- **Token Staking**: Users can stake supported tokens.
- **Reward Distribution**: Periodic rewards are calculated and distributed based on staking amounts and time.
- **Emergency Withdrawals**: Provides the owner with the ability to withdraw all funds during emergencies.
- **Pausable and Ownable**: Implements pause functionality and ownership control for enhanced security.
- **Flexible Configuration**: Provides a pluggable configuration scheme, allowing customization of interest rates, support for different tokens, and setting the waiting time between user withdrawal requests and successful withdrawals.
- **Comprehensive Query Mechanism**: Offers functionalities such as reward calculation, staking amount queries, and querying withdrawal availability times.
- **Efficient Execution Mechanism**: Features standardized and efficient code that minimizes gas consumption while maintaining readability.

Generally, the *Vault* will operate on EVM-based networks and support stablecoins such as *USDT* and *USDC*, providing users with secure, convenient, and fast staking and withdrawal services. The *Vault*'s administrator will ensure the system's reliability and maintain a bot to respond promptly to user actions.

2.2 Workflow

From *Figure 1*, the overall operational flow of the *Vault* system is demonstrated. A complete process is illustrated through a case study, from staking to completing a claim, explaining the actions of three different roles: **Owner**, **User** and **Bot**.

1. **Owner** deploys the *Vault* system and initializes it by setting the basic configurations.
2. **User** stakes tokens supported by the *Vault*. They can perform multiple staking operations repeatedly.
3. The **Bot** listens for staking events from users and reallocates the assets in the *Vault* according to the configured strategies. For instance, it transfers the liquidity provided by users to *Ceffu* to gain potential value-added services.
4. At some point in the future, **User** submits a withdrawal request to the *Vault*. They then need to wait for a **14-day** buffer period. Any portion of the position not withdrawn will continue to generate rewards.

5. The **Bot** listens for user withdrawal requests and reallocates assets within the **14-day** period. By default, the Bot retrieves funds from *Ceffu*, but theoretically, any source is permissible.

6. After the **14-day** waiting period, **User** can successfully complete their withdrawal requests.

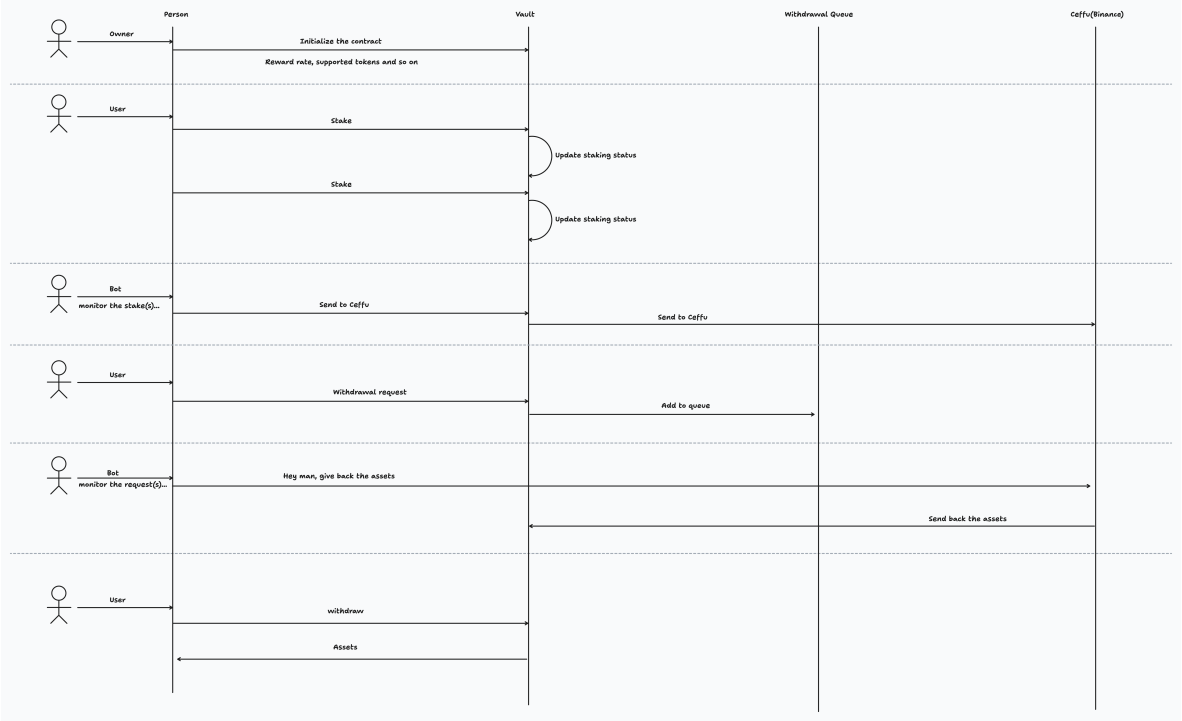

Figure 1: Workflow

3 Rate Theory

3.1 Reward Rate Model

The assets staked by users will be calculated based on a simple reward rate model, with rewards accumulating per second. Generally, the *Vault* system has the following conventions regarding the reward rate model and its implementation:

- After the first staking, the **User** will have a position in the system. Users can stake at any time, but the staking amount is subject to upper and lower limits.
- Once a **User** submits a claim request, they can not modify it and must wait for a buffer period (default is 14 days). After the buffer period ends, the **User** can proceed with the claim to receive the tokens. And once the **User** submits the claim request, the portion of the position requested for withdrawal will no longer generate rewards (the remaining portion will continue to generate rewards). **User** can create another request as long as he has claimable assets.
- The maximum amount a **User** can request for withdrawal is the sum of the principal and rewards. The withdrawal amount is prioritized from rewards; if the rewards are insufficient, the principal will be used to supplement the amount. The principal used for supplementation will no longer generate rewards.
- After the **14-day** waiting period, users can withdraw the requested amount from the *Vault*. Even if the **User** does not withdraw, this amount will not be treated as a position and will not generate rewards.

Due to the potential changes in the reward rate, there are two possible methods for calculating the total rewards:

- *Figure 2*: If the reward rate remains unchanged, the reward is accumulated directly.
- *Figure 3*: If the reward rate changes, the reward is accumulated in segments.

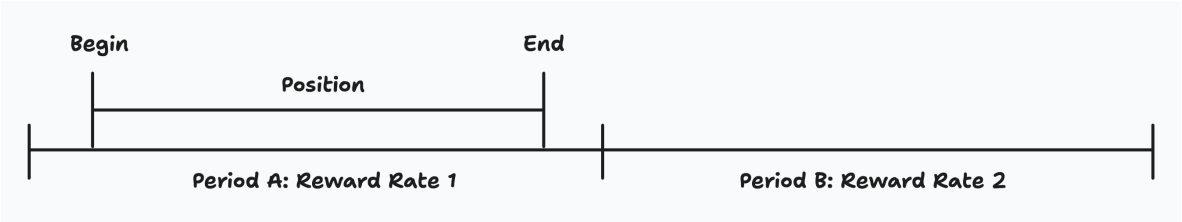

Figure 2: Scenario 1

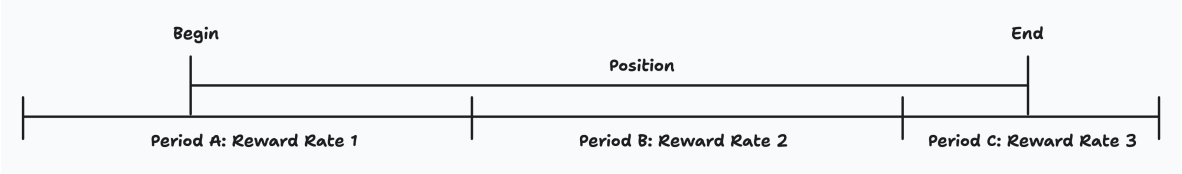

Figure 3: Scenario 2

We quantify the above scenarios into a mathematical formula. Assume that the time difference between *Begin* and *End* is *elapsedTime*, and there are n periods with different reward rates during this time. The staked amount held at position is represented as *stakedAmount*, with each period having a different Reward Rate (APR). Assuming the total reward is *f(x)*, we can derive::

$$f(x) = \sum_{k=1}^{n} \left(\text{stakedAmount} \cdot \text{APR}_k \cdot \frac{\text{elapsedTime}_k}{\text{ONE\_YEAR}} \right), \quad n \in \{1, 2, 3, 4, \dots\}$$

It is worth noting that in the *Vault* system, `ONE_YEAR` is 365 days plus 0.25 days. This accounts for the extra day added due to leap years.

3.2 Practical Case

The *Figure 4* illustrates the process of a user depositing, submitting a claim request, and completing the claim in the *Vault* system. And *Table 1* shows the accumulation of rewards over different periods.

Rewards	Total Rewards	Total Stake
a: from 100	a	100
b: from 300	a + b	300
c: from 300	a + b + c	300
d: from 600	a + b + c + d	600
e: from 600	a + b + c + d + e	600
f: from 600	a + b + c + d + e + f	600

Table 1: Reward Accumulation

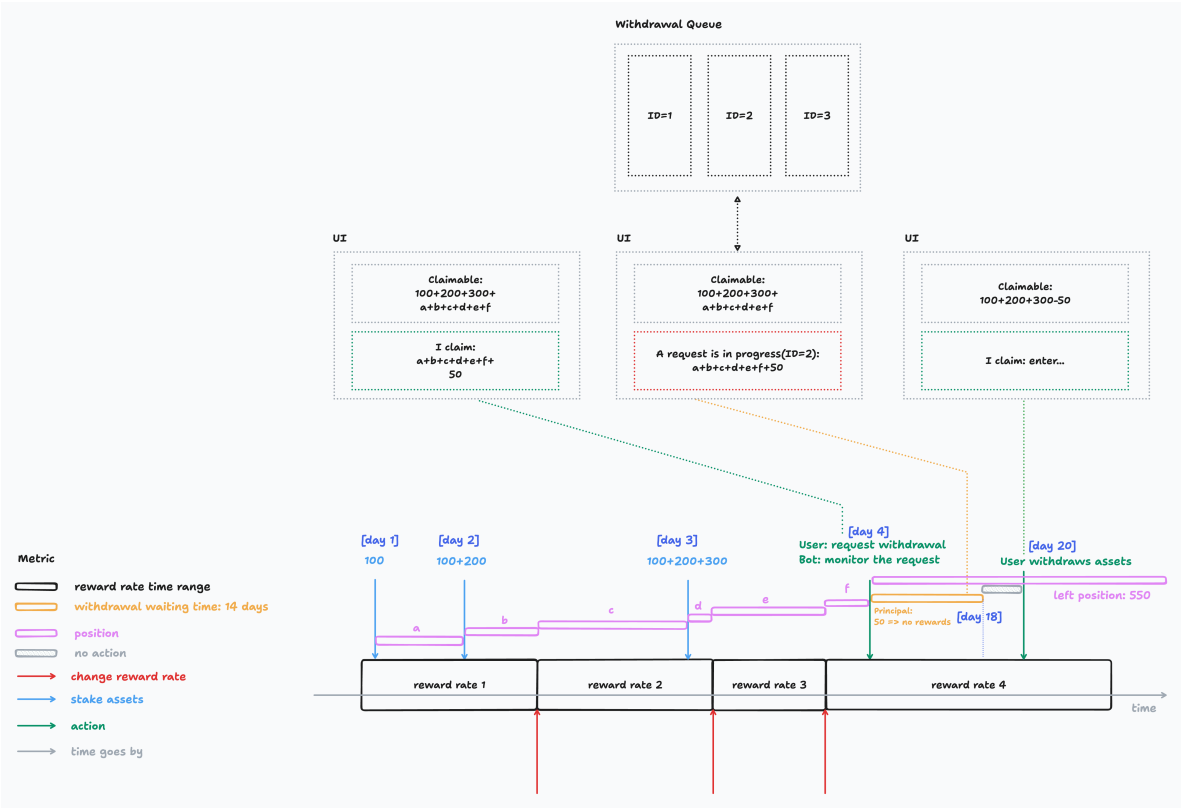

Figure 4: Practical Demo

4 Interface

4.1 Write

- **stake_66380860(address _token, uint256 _stakedAmount)**: Allows a user to stake a specified amount of a supported token. Once called, the specified amount of the token is transferred to the contract and marked as staked.
- **requestClaim_8135334(address _token, uint256 _amount)**: Enables a user to request a claim for a specified amount of a supported token, and returns a request id. The user will then need to wait for a buffer period. If the requested withdrawal amount is equal to the maximum value of *uint256*, it is considered as withdrawing all assets of the user.
- **claim_41202704(uint256 _queueID)**: After the user has completed the buffer period, they can call this function to claim.
- **transferToCeffu(address _token, uint256 _amount)**: Transfers a specified amount of a token to the ceffu address. Only the bot could call it.
- **emergencyWithdraw(address _token, address _receiver)**: Allows for emergency withdrawal of tokens from the contract to a specified receiver address. Only the owner could call it.
- **addSupportedToken(address _token, uint256 _minAmount, uint256 _maxAmount)**: Adds a new ERC20 token to the list of supported tokens and sets its staking limits. The function allows setting a minimum and maximum stake amount for the token. Only the owner could call it.

- **setStakeLimit(address _token, uint256 _minAmount, uint256 _maxAmount)**: Sets the minimum and maximum staking limits for a specific token. This function allows adjusting the staking amount range for each supported token. Only owner could call it.
- **setRewardRate(address _token, uint256 _newRewardRate)**: Updates the reward rate for a specific token. Only the owner could call it.
- **setCeffu(address _newCeffu)**: Updates the ceffu address. Only the owner could call it.
- **setWaitingTime(uint256 _newWaitingTime)**: Changes the waiting time for staking or reward claims. Only the owner could call it.

4.2 Read

- **getClaimableRewardsWithTargetTime(address _user, address _token, uint256 _targetTime)**: This function returns the amount of rewards a user can claim for a specified token at a given target time. It calculates the potential rewards based on the user's staking history and the target time.
- **getClaimableAssets(address _user, address _token)**: Returns the total amount of claimable assets (Principal + Rewards) for a specified user and token.
- **getStakedAmount(address _user, address _token)**: Returns the total amount of a specified token staked by a user.
- **getContractBalance(address _token)**: Returns the contract's balance of a specified token.
- **getStakeHistory(address _user, address _token, uint256 _index)**: Returns a user's stake history for a specified token at a given index. This function retrieves historical staking records, such as the amount and time of staking, allowing users to track their past staking activities.
- **getClaimHistory(address _user, address _token, uint256 _index)**: Returns a user's claim history for a specified token at a given index. This function retrieves information about past claims, such as the claim amount and time, helping users to track their claimed rewards.
- **getCurrentRewardRate(address _token)**: Returns the current reward rate for a specified token.
- **getClaimQueueInfo(uint256 _index)**: Returns the details of a claim queue item at a specified index.
- **getClaimQueueIDs(address _user, address _token)**: Returns the request information for a user and a specific token.
- **getClaimableRewards(address _user, address _token)**: Returns the rewards that user could claim.
- **getStakeHistoryLength(address _user, address _token)**: Returns the staking history length.
- **getClaimHistoryLength(address _user, address _token)**: Returns the claim history length.
- **getTotalRewards(address _user, address _token)**: Retrieve the total rewards earned by the user historically and the combined amount currently claimable.
- **getTVL()**: Returns the TVL.

Name	Version
Machine	MacOS (Sonoma 14.4), Memory 18GB
Forge	forge 0.2.0 (9501589 2024-10-30T00:25:46.926240000Z)
Solidity	0.8.28 (cancun)

Table 2: Configurations

4.3 Gas Report

The configuration is shown in *Table 2*

We conducted tests under the Foundry framework and obtained the following benchmark results:

Function Name	min	avg	median	max	# calls
getTVL	392	392	392	392	2
WAITING_TIME	417	417	417	417	1
ceffu	470	470	470	470	1
minStakeAmount	597	597	597	597	1
supportedTokens	599	599	599	599	1
maxStakeAmount	642	642	642	642	1
getCurrentRewardRate	1513	1513	1513	1513	1
getContractBalance	2066	2066	2066	2066	1
getClaimQueueInfo	1923	1923	1923	1923	3
getClaimQueueIDs	3711	4320	4320	4929	2
getStakeHistory	4196	4751	4751	5307	2
getClaimHistory	4525	5266	5637	5637	3
getStakedAmount	3484	5687	6789	6789	3
getClaimableAssets	7948	13688	14559	18559	3
getTotalRewards	10394	10394	10394	10394	1
getClaimableRewardsWithTargetTime	9177	9739	9739	10301	2
setCeffu	26946	30389	30389	33832	2
setWaitingTime	30407	30407	30407	30407	1
emergencyWithdraw	59054	59054	59054	59054	1
setStakeLimit	56507	56507	56507	56507	1
transferToCeffu	59899	59899	59899	59899	1
addSupportedToken	126590	126590	126590	126590	1
setRewardRate	107051	107051	107051	107051	1
requestClaim_8135334	64111	236449	271889	284190	7
stake_66380860	203165	238820	245479	255079	8
claim_41202704	197948	245098	252048	271401	5

Table 3: Function Gas Usage Details (Sorted by Average)

5 Test

5.1 Local

We test the *Vault* system under the Foundry testing framework. The results are shown in *Table 4*.

Function	Gas Consumption	Status
testSetWaitingTime()	21168	PASS
testSetStakeLimit()	50782	PASS
testEmergencyWithdraw()	79487	PASS
testSetAndTransferToCeffu()	95910	PASS
testAddSupportedToken()	117101	PASS
testStake()	299259	PASS
testRequestClaim()	460682	PASS
testClaimAssets_Scenario_A()	643631	PASS
testClaimAssets_Scenario_B()	767435	PASS
testClaimAssets_Scenario_C()	813805	PASS
testClaimAssets_Scenario_D()	955021	PASS

Table 4: Local Testing Results

5.2 Sepolia Testing Network

We deployed on Ethereum's Sepolia testing network for product testing, and the following information is available:

- **Mock ERC20 address**: 0x4F406253d32792a847516Df1d33F0aF88FEE15A5
- **Vault address**: 0x493B4F8415a659f0f3CEF84A5923835adae24BB9
- **Owner address**: 0x00004d81A8403D09D14851b7EdEEcF89ECB38563
- **Owner private key**: 0x8a06963ac66a645a61fe54b77f89c8401c2938a84e5f5430c0eb8677d14228e8

6 Summary

The *Vault* system by Zerobase sets a strong foundation for secure and efficient decentralized staking solutions. With its robust design and user-centric approach, it demonstrates the potential of blockchain technology in transforming financial services. The system aims to incorporate multi-chain compatibility, adaptive reward mechanisms, and enhanced analytics to address evolving user needs and market dynamics.

Copyright and Usage Disclaimer